

**18CSC204J – Algorithm Analysis and Design**

**Record Work**

**Registration Number:**

**RA2011033010053**

**RA2011033010026**

**RA2011033010031**

**Name of the Student:**

**Aniket Bada Panda**

**Siddharth Pandey**

**Pratham Sahu**

**Semester / Year: IV/2nd year**

**Department:** **CINTEL (Computational Intelligence)**

## CONTRIBUTION TABLE

| 1. | Problem Definition |
| --- | --- |
| 2. | Problem Explanation with diagram and example |
| 3. | Design Techniques used (Divide and conquer, Greedy method,  Dynamic programming, Backtracking, Branch and bound or  Randomized algorithm)  - Briefly explain the general technique |
| 4. | Algorithm for the problem |
| 5. | Explanation of algorithm with example |
| 6. | Complexity analysis |
| 7. | Conclusion |
| 8. | References |

**BIN PACKING PROBLEM USING GREEDY APPROACH**

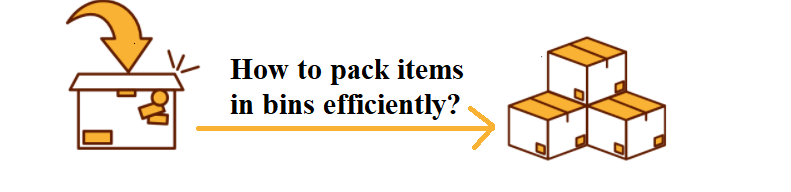
1. **Problem Definition:**

Bin Packing Problem can be solved using Greedy approach.

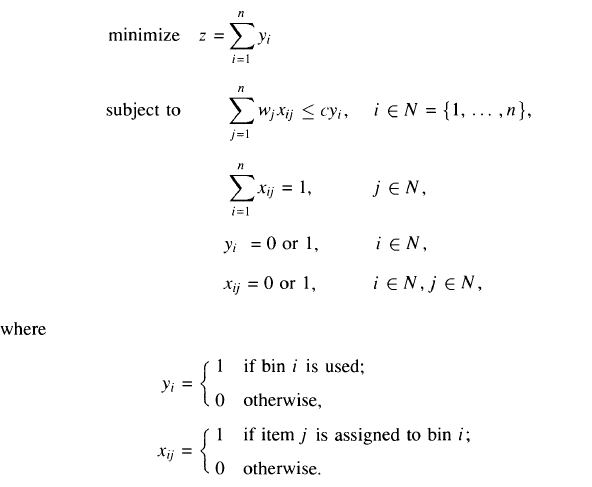
Given n items of different weights and bins each of capacity c, assign each item to a bin such that number of total used bins is minimized. It may be assumed that all items have weights smaller than bin capacity.

1. **Problem Explanation with diagram and example:**

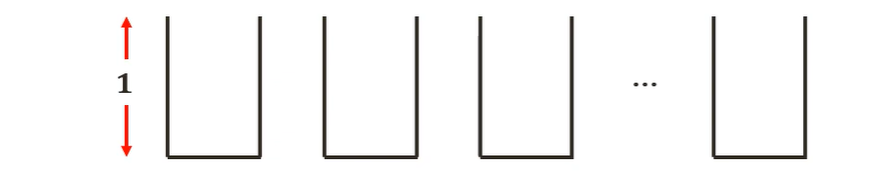
Bin Packing problem involves assigning n items of different weights and bins each of capacity c to a bin such that number of total used bins is minimized. It may be assumed that all items have weights smaller than bin capacity.



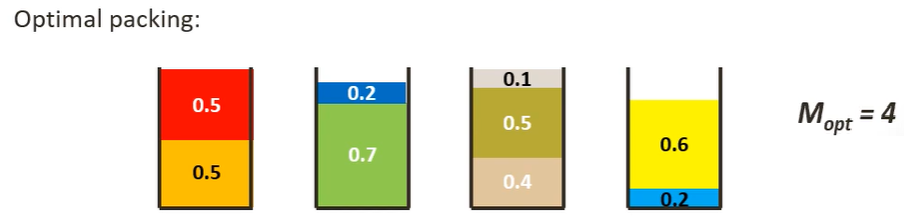
A possible mathematical formulation of the problem is -



As you can see, we have a broad rule of approximation and not an exact algorithm. Such algorithms are called [NP problems](https://iq.opengenus.org/approximate-algorithms-for-np-problems/). In fact, Bin Packing Problem is a [NP-hard problem](https://iq.opengenus.org/np-hard/).

For the Bin-Packing problem, let us consider bins of size 1-   


Assuming the sizes of the items be {0.5, 0.7, 0.5, 0.2, 0.4, 0.2, 0.5, 0.1, 0.6}.  
The most optimal solution (z(I))for this instance I would be:



**3. Design Technique:**

A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment. It doesn't worry whether the current best result will bring the overall optimal result.

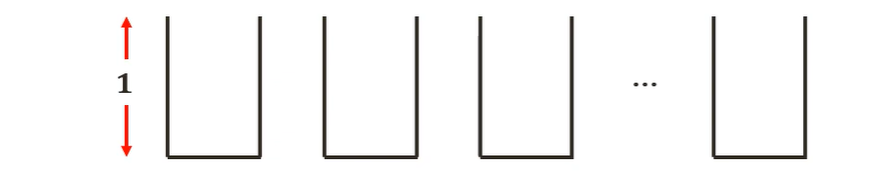
The algorithm never reverses the earlier decision even if the choice is wrong. It works in a top-down approach.

This algorithm may not produce the best result for all the problems. It's because it always goes for the local best choice to produce the global best result.

# 1) Next Fit algorithm -

The simplest approximate approach to the bin packing problem is the Next-Fit (NF) algorithm which is explained later in this article. The first item is assigned to bin 1. Items 2,... ,n are then considered by increasing indices : each item is assigned to the current bin, if it fits; otherwise, it is assigned to a new bin, which becomes the current one.

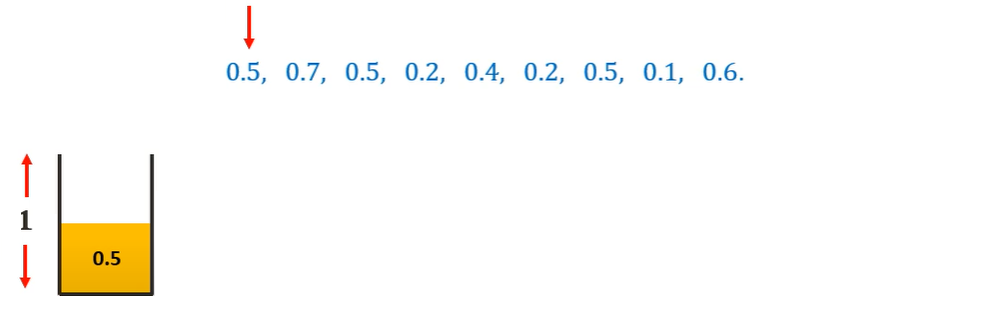
Visual Representation

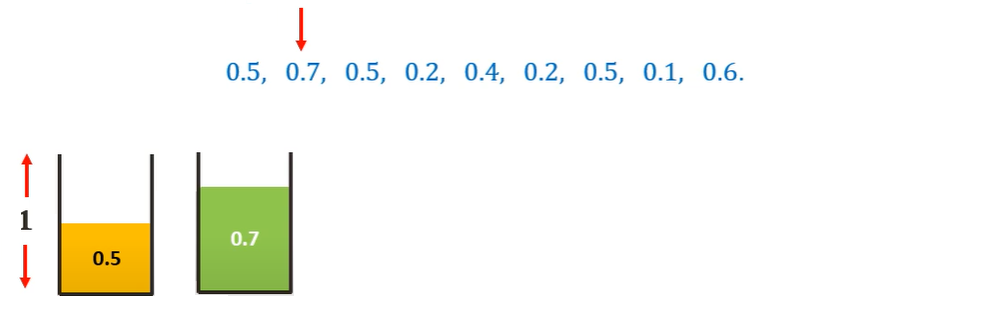
Let us consider the same example as used above and bins of size 1  


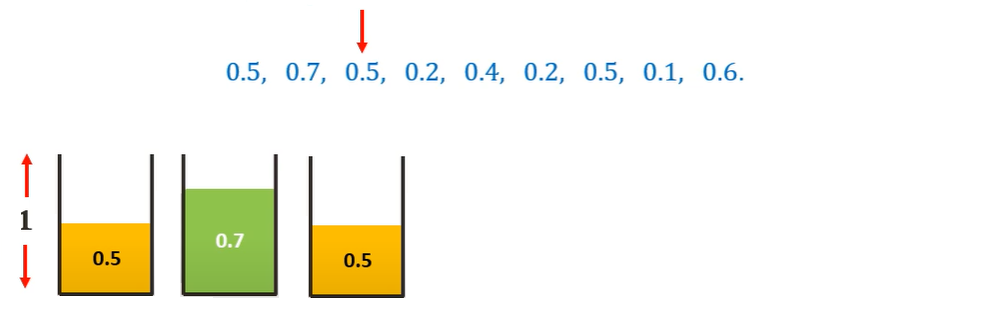
Assuming the sizes of the items be {0.5, 0.7, 0.5, 0.2, 0.4, 0.2, 0.5, 0.1, 0.6}.

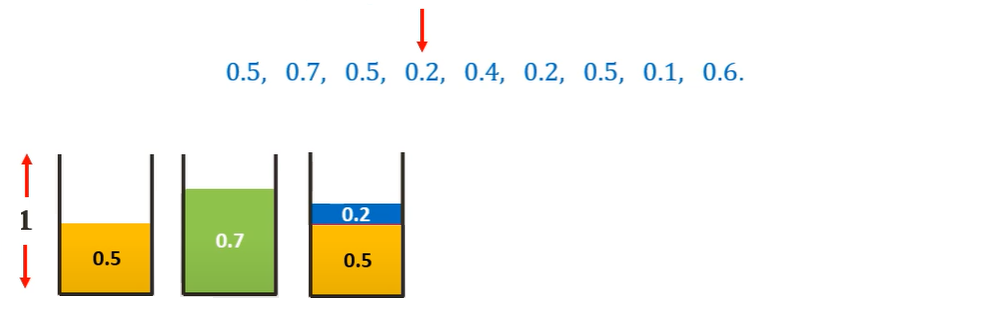
The minimum number of bins required would be Ceil ((Total Weight) / (Bin Capacity)) =  
Ceil(3.7/1) = 4 bins.

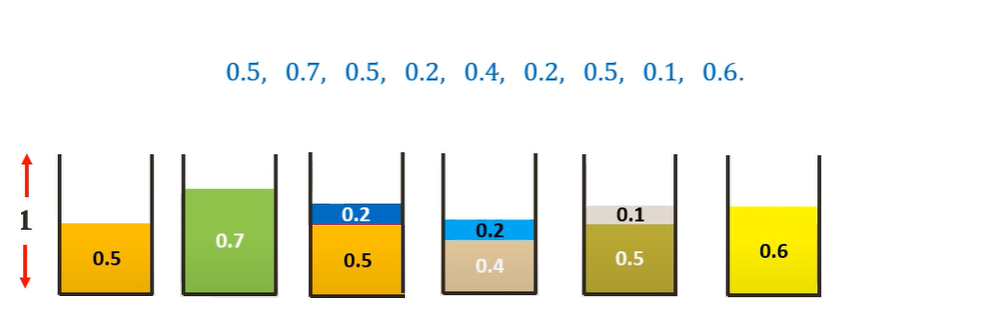
The Next fit solution (NF(I)) for this instance I would be-

Considering 0.5 sized item first, we can place it in the first bin  


Moving on to the 0.7 sized item, we cannot place it in the first bin. Hence we place it in a new bin.  


Moving on to the 0.5 sized item, we cannot place it in the current bin. Hence we place it in a new bin.  


Moving on to the 0.2 sized item, we can place it in the current (third bin)  


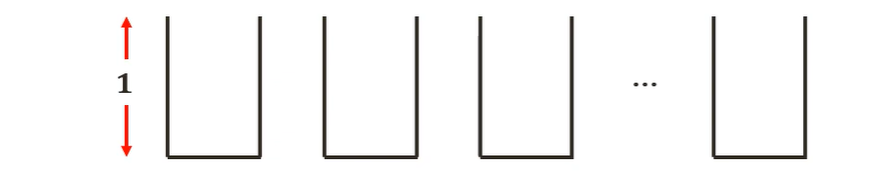
Similarly, placing all the other items following the Next-Fit algorithm we get-  


Thus, we need 6 bins as opposed to the 4 bins of the optimal solution. We can see that this algorithm is not very efficient.

# 2) First Fit algorithm -

A better algorithm, First-Fit (FF), considers the items according to increasing  
indices and assigns each item to the lowest indexed initialized bin into which it  
fits; only when the current item cannot fit into any initialized bin, is a new bin  
introduced

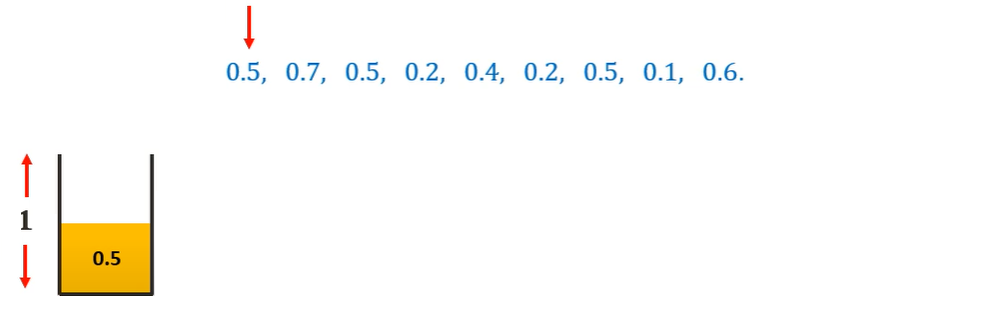
Visual Representation

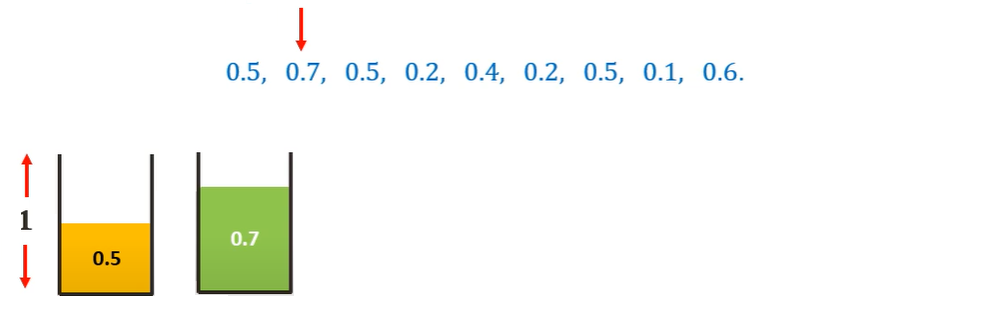
Let us consider the same example as used above and bins of size 1  


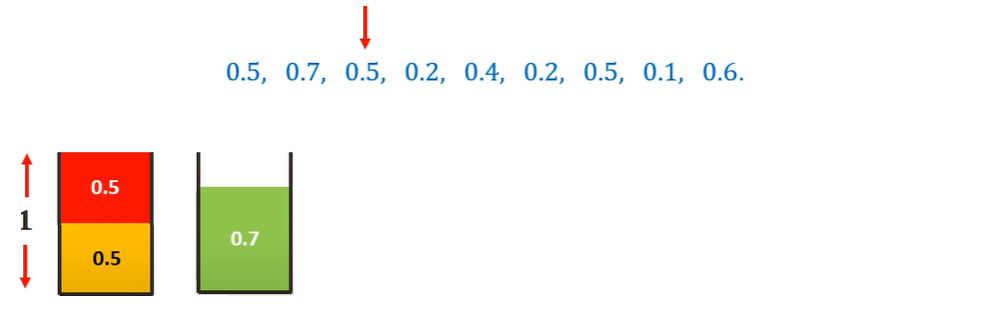
Assuming the sizes of the items be {0.5, 0.7, 0.5, 0.2, 0.4, 0.2, 0.5, 0.1, 0.6}.

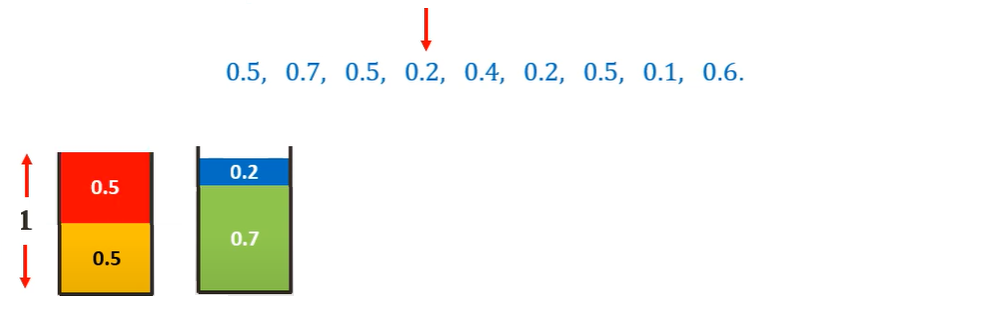
The minimum number of bins required would be Ceil ((Total Weight) / (Bin Capacity)) =  
Ceil(3.7/1) = 4 bins.

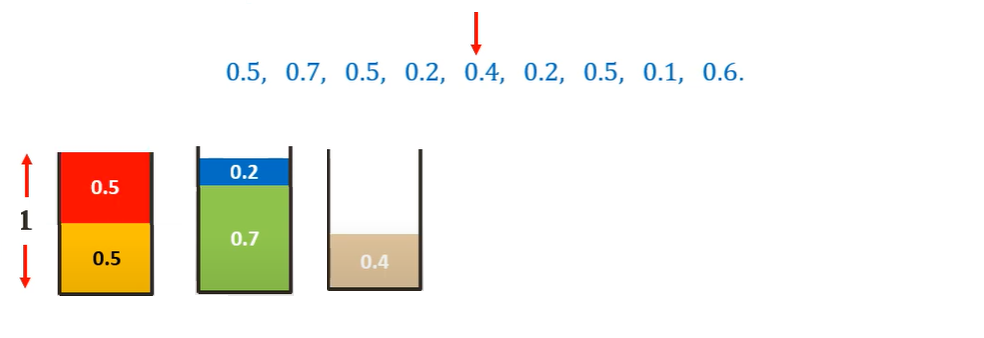
The First fit solution (FF(I)) for this instance I would be-

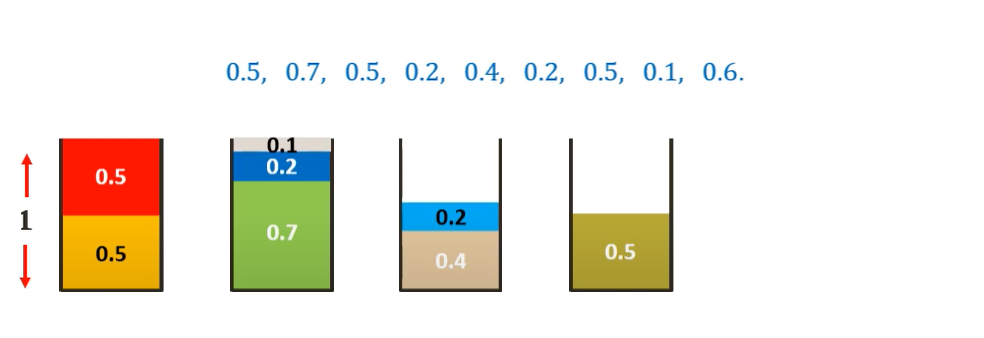
Considering 0.5 sized item first, we can place it in the first bin  


Moving on to the 0.7 sized item, we cannot place it in the first bin. Hence, we place it in a new bin.  


Moving on to the 0.5 sized item, we can place it in the first bin.  


Moving on to the 0.2 sized item, we can place it in the first bin, we check with the second bin and we can place it there.  


Moving on to the 0.4 sized item, we cannot place it in any existing bin. Hence we place it in a new bin.  


Similarly, placing all the other items following the First-Fit algorithm we get-  


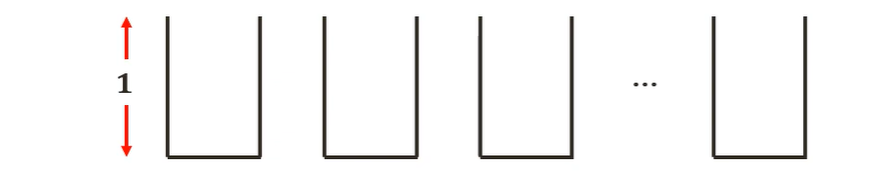
Thus, we need 5 bins as opposed to the 4 bins of the optimal solution but is much more efficient than Next-Fit algorithm.

# 3) Best Fit Algorithm -

The next algorithm, Best-Fit (BF), is obtained from FF by assigning the current  
item to the feasible bin (if any) having the smallest residual capacity (breaking  
ties in favour of the lowest indexed bin).

Simply put, the idea is to places the next item in the *tightest* spot. That is, put it in the bin so that the smallest empty space is left.

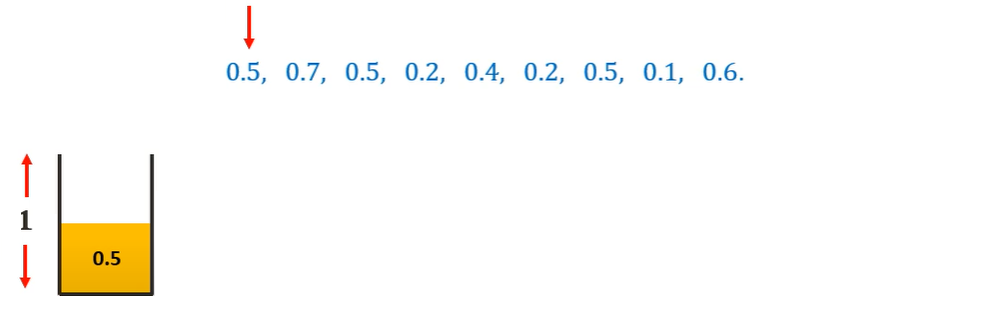
Visual Representation

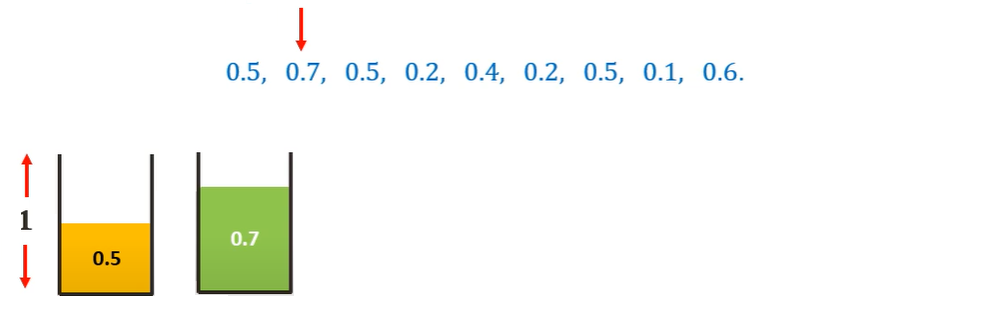
Let us consider the same example as used above and bins of size 1  


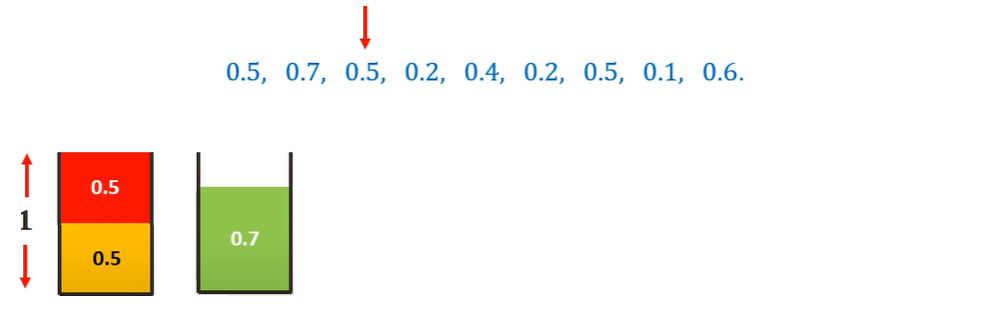
Assuming the sizes of the items be {0.5, 0.7, 0.5, 0.2, 0.4, 0.2, 0.5, 0.1, 0.6}.

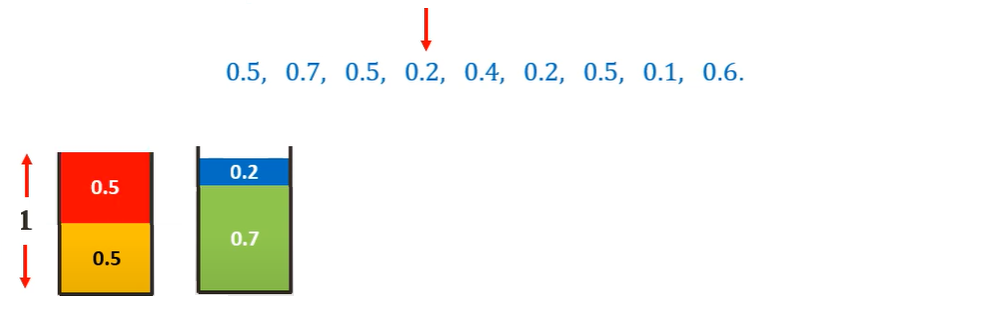
The minimum number of bins required would be Ceil ((Total Weight) / (Bin Capacity)) =  
Ceil(3.7/1) = 4 bins.

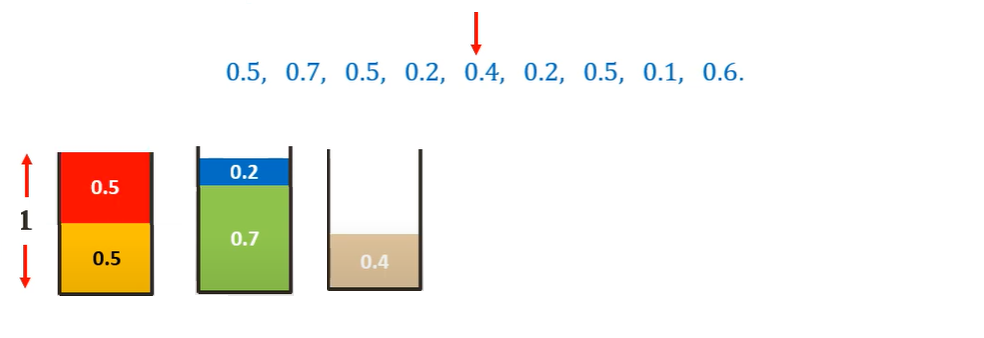
The First fit solution (FF(I)) for this instance I would be-

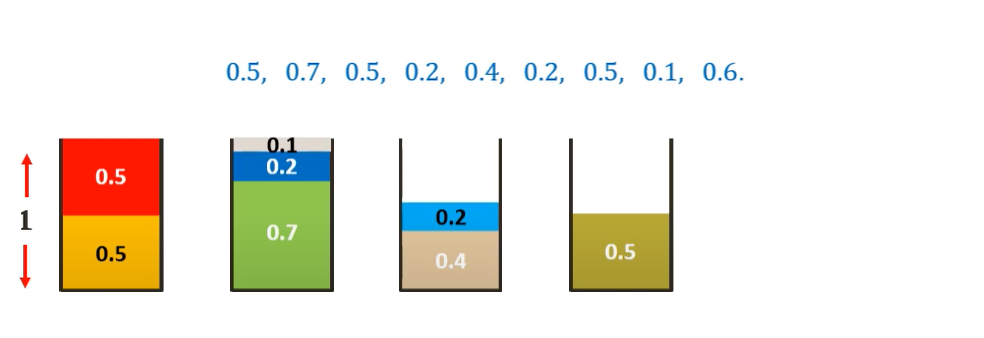
Considering 0.5 sized item first, we can place it in the first bin  


Moving on to the 0.7 sized item, we cannot place it in the first bin. Hence, we place it in a new bin.  


Moving on to the 0.5 sized item, we can place it in the first bin tightly.  


Moving on to the 0.2 sized item, we cannot place it in the first bin but we can place it in second bin tightly.  


Moving on to the 0.4 sized item, we cannot place it in any existing bin. Hence we place it in a new bin.  


Similarly, placing all the other items following the First-Fit algorithm we get-  


Thus we need 5 bins as opposed to the 4 bins of the optimal solution but is much more efficient than Next-Fit algorithm.

# 4) Worst Fit Algorithm -

This algorithm involves an idea to places the next item in the least tight spot to even out the bins. In other words, put it in the bin so that most empty space is left.

1. **Algorithm for the problem:**

**Greedy Algorithm:** Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. It doesn't worry whether the current best result will bring the overall optimal result.

So, the problems where choosing locally optimal also leads to global solution are the best fit for Greedy. A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment.

1. **Explanation of algorithm with an example:**

**Approach:**

There are several ways implement Bin Packing as discussed earlier. One of the several methods in Best-Fit Algorithm.

**Best Fit:**   
The idea is to places the next item in the \*tightest\* spot. That is, put it in the bin so that the smallest empty space is left.

**Algorithm:**

1. Declare a function which takes: the array of weights, maximum value of bins (equal to number of weights) and capacity of each bin.
2. In the function, initialize a variable – result to value 0.
3. Declare an array to store bins that are created.
4. Run a for loop: i 🡨 0 to n-1:
5. Initialize a variable(j) that would store value of the best bin that can accommodate weight[i].
6. Run another loop: j 🡨 0 to n-1 to find the best bin to accommodate weight[i] by comparing the difference of capacity and weight with a minimum value.
7. In case, there is no result, create a new bin and increment the count.
8. Return count.
9. **Complexity analysis:**

* Worst case time complexity: **O(n\*n)**
* Average case time complexity: **O(n\*n)**
* Best case time complexity (Can be achieved using Self-balancing Binary trees): **O(nlogn)**
* Space complexity: **O(n)**

1. **Conclusion:**

Minimum number of bins required to store a number of weights has been determined.

1. **References:**

* [**https://www.geeksforgeeks.org/bin-packing-problem-minimize-number-of-used-bins/**](https://www.geeksforgeeks.org/bin-packing-problem-minimize-number-of-used-bins/)
* **https://www.geeksforgeeks.org/greedy-algorithms/**